Information Systems 38 (2013) 908–926

Contents lists available at SciVerse ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

MFIBlocks: An effective blocking algorithm for entity resolution

Batya Kenig*, Avigdor Gal

Technion-Israel Institute of Technology, Haifa, Israel

ARTICLE INFO

Available online 2 December 2012

Keywords:
Entity resolution
Blocking

ABSTRACT

Entity resolution is the process of discovering groups of tuples that correspond to the same real-world entity. Blocking algorithms separate tuples into blocks that are likely to contain matching pairs. Tuning is a major challenge in the blocking process and in particular, high expertise is needed in contemporary blocking algorithms to construct a *blocking key*, based on which tuples are assigned to blocks. In this work, we introduce a blocking approach that avoids selecting a blocking key altogether, relieving the user from this difficult task. The approach is based on maximal frequent itemsets selection, allowing early evaluation of block quality based on the overall commonality of its members. A unique feature of the proposed algorithm is the use of prior knowledge of the estimated size of duplicate sets in enhancing the blocking accuracy. We report on a thorough empirical analysis, using common benchmarks of both real-world and synthetic datasets to exhibit the effectiveness and efficiency of our approach.

1. Introduction

Entity resolution is a fundamental problem in data integration. It refers to the problem of determining which tuples (using relational notation) resolve to the same real-world entity. At the heart of the entity resolution process is the challenge to match tuples that share no unique identifiers, may come from non-matching schemata, and may consist of typos and out-of-date or missing information. Entity resolution algorithms typically compare the content of tuples to determine if they match and merge matching tuples into one. Such a comparison may be prohibitive for big datasets if all tuple pairs are compared and hence pairwise comparison is typically preceded by a blocking phase, a procedure that divides tuples into mutually exclusive subsets called *blocks*.

Tuples assigned to the same block are ideally resolved to the same real-world entity. In practice, tuples in a block are all candidates for the more rigorous tuple pair-wise comparison. Therefore, blocking algorithms should be designed to produce quality blocks, containing as many tuple matches and avoiding as many non-matches as possible. Balancing the two requirements calls for an optimal block size that should not be too small, to avoid false negatives, or too large, to avoid false positives. Larger blocks also increase the time spent on pair-wise tuple comparison and hence, blocking algorithms aim at balancing the need to reduce false negatives and the need to reduce performance overhead.

Several blocking algorithms were proposed in the literature, *e.g.*, sorted neighborhood [1], canopy clustering [2], and *q*-gram indexing [3]. In this work we offer a novel blocking algorithm, dubbed MFIBlocks, that is based on iteratively applying an algorithm for mining Maximal Frequent Itemsets [4]. MFIBlocks offers four major unique features. Firstly, MFIBlocks waives the need to manually design a *blocking key*, the value of one or more of a tuple's attributes. Blocking keys in contemporary blocking algorithms have to be carefully designed to avoid false negatives by assigning matching tuples to different blocks. Therefore, attributes in the blocking key should

* Corresponding author. Tel.: +972 528461692
E-mail addresses: batyak@tx.technion.ac.il, batya.kenig@gmail.com (B. Kenig), avigal@ie.technion.ac.il (A. Gal).

0306-4379/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2012.11.008

contain few errors and missing values and the design of a blocking key should take into account the frequency distribution of values in the attributes of the blocking key to balance block sizes. MFIBlocks relieves the designer from the difficult task of constructing a blocking key.

Second, MFIBlocks localizes the search for similar tuples and is able to uncover blocks of tuples that are similar in multiple, possibly overlapping sets of attributes. MFIBlocks allows a dynamic, automatic, and flexible selection of a blocking key, so that different blocks can be created based on different keys. This approach is in line with the state-of-the-art in clustering literature (see, *e.g.*, [5]) and extends the current perception of a single-key-fits-all .

Blocks, created by the algorithm, are constrained to satisfy the *compact set* (CS) and *sparse neighborhood* (SN) [6] properties. As such, local structural properties of the dataset are used during the discovery and evaluation of tuple clusters and the number of comparisons for each tuple is kept low, even though the same tuple can appear in several clusters (using multiple keys) simultaneously.

Finally, MFIBlocks is designed to discover entity sets of matching tuples with largely varying sizes. MFIBlocks effectively utilizes a-priori knowledge of the sizes of matching entity sets, by discovering clusters of the appropriate size having the largest possible commonality.

This paper introduces the MFIBlocks algorithm, discusses its properties, and presents a thorough empirical analysis, demonstrating its superior effectiveness. We offer techniques to make the execution time performance of the algorithm attractive, balancing execution time with effectiveness. The novelty of our paper is as follows:

- We present a novel blocking algorithm that reduces the effort of manual tuning and enables locating clusters of similar tuples in multiple, possibly overlapping sets of attributes.
- We provide a thorough empirical analysis of the algorithm performance, using both real-world and synthetic datasets, and show its superior effectiveness over common benchmarks.
- We offer methods to ensure the efficiency of the algorithm, demonstrating the trade-off between execution time and effectiveness.

The rest of the paper is organized as follows. Section 2 provides a brief overview of the entity resolution process and frequent itemset mining. The building blocks of the proposed approach are provided in Section 3. The algorithm is presented in Section 4. In Section 5 we provide an empirical analysis over several benchmark datasets. Section 6 provides an overview of related work, positioning our work on this background. We conclude in Section 7 with a summary and a discussion of future work.

2. Preliminaries

The general, unsupervised entity resolution process illustrated in Fig. 1 contains blocking, comparison and classification stages. Occasionally, a standardization process precedes these steps to increase the process effectiveness.

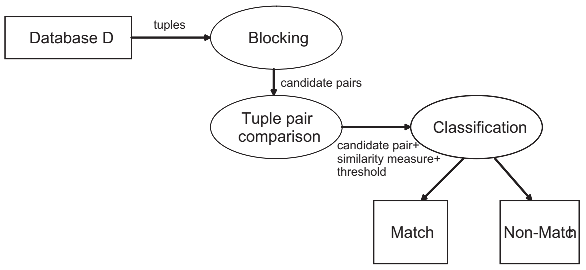

Fig. 1. Entity resolution process.

Table 1
Sample transaction database.

Transaction	Items
t_1	a_1,b_1,c_1
t_2	a_1,b_1,c_2
t_3	a_2,b_2,c_3
t_4	a_2,b_2,c_2
t_5	a_1,b_3,c_4

The output of the blocking stage is a set of blocks where each pair of tuples in a block is considered a *candidate pair*. Only candidate pairs are then compared and other pairs are automatically classified as non-matches .

Using similarity measures such as the Jaccard coefficient [7], candidate pairs are classified as matching or non-matching . A tuple pair (t_1,t_2), over a schema of k comparable attributes, is represented as a vector $v=[v_1,\dots,v_k]$. Each v_i is a measure of the similarity of the i-th attribute. In most cases, the entries in the vector v are in the range [0,1]. A function f over the values of these entries is used to classify a pair according to a predefined threshold.

In this paper we focus on the blocking stage and suggest using maximal frequent itemsets to generate blocks of tuples as candidates for the classification stage. For completeness sake, we now provide an overview of the concepts of frequent and maximal frequent itemsets [8]. Frequent itemsets originated from the data mining field and was used in other fields as well, *e.g.*, for identifying similar Web documents [9]. This overview uses the notions of items and transactions, based on the native vocabulary of data mining. It is worth noting that the term *transaction* has a different (albeit related) meaning to the same term in the database literature, referring to a bag of items from a raw dataset, *e.g.*, billing transactions. Let $M=\{I_1,I_2,\dots,I_m\}$ be a set of items and let $T=\langle T_1,T_2,\cdots,T_n\rangle$ be a set of transactions. A transaction $T_i=(tid,I)$ with identifier *tid* contains a set of items $I\subseteq M$. The *support* of some set of items $I\subseteq M$ is the set of transactions in T that contain all items in I. Each transaction possibly contains additional items.

Example 1. Table 1 contains five transactions with the items $M=\{a_1,a_2,b_1,b_2,b_3,c_1,c_2,c_3,c_4\}$. Transaction t_1